Amendments to the Specification

Please replace the paragraph beginning at page 37, line 9, with the following rewritten paragraph:

With reference now to Figure 6A, it is assumed that two classes, say classes C1 and C2 loaded by class loaders L1 and L2 respectively both reference a class "Fred", with L1 as the requesting loader and L2 as the providing loader. This leads to the requirement for a type constraint, which is typically detected as the system attempts to call the relevant method in C1 that triggers the constraint. Assuming for the moment that after searching for Fred in the internal caches 610 and 611 of ~~both~~ L1 and L2, respectively, neither has yet resolved Fred. Therefore, the system creates a so-called dummy entry in the internal ~~cache~~ caches for each of L1 and L2 ("dummy" because it does not yet actually identify the true location of Fred). For L2 the dummy entry 621 simply contains the identifier "Fred" in the class label field (see 522 in Figure 5). The corresponding class label field 630 in the constraint data structure 620 in internal cache ~~604~~ 610 for Figure 6A likewise contains the value Fred. However, data structure also contains an additional value in the constraint parent field 620, which points to the data structure 621 for that constraint in the internal cache 611 of class loader L2. Notice therefore that this reflects the asymmetric relationship between a requesting class loader (L1) and a providing class loader (L2), and that L2 must be a direct or indirect parent of L1.

Please replace the paragraph beginning at page 40, line 14, with the following rewritten paragraph:

An alternative possibility is that when L1 comes to resolve Fred, that this class has already been resolved by L2. This situation

2

is illustrated in Figure 6D, which is updated from Figure 6A in that class Fred is now loaded 699, and the class reference 626 in the internal cache 611 for loader L2 has been altered to point to Fred. Now when L1 tries to resolve Fred, it must ensure that it obtains the same class reference as stored within field 626 within the internal cache for L2, otherwise the constraint is violated. Note that L1 knows to perform this check because the constraint parent field 640 is set for its own cache entry for Fred 620, and points to the cache entry for Fred for L2. Assuming that the correct class reference is obtained by L1, ~~the~~ <u>then</u> it can update its own internal cache accordingly, which would lead again to the end result of Figure 6C (which represents the end result for all successful scenarios).

Please replace the paragraph beginning at page 41, line 15, with the following rewritten paragraph:

The above processing is illustrated in the flowcharts of Figure 7 and Figure 8. The processing of Figure 7 is triggered by the identification of a type constraint between class loaders L1 and L2 in respect of class C (step 705). Thus a test is first made to see if L1 has already resolved C (step 710), and if so, a further test is made to see if L2 has already resolved C (step 715). If this is again positive, the class references obtained by L1 and L2 for C are compared to ensure that they are identical (step 725). If they match, then the method successfully exits (step 735), otherwise a constraint violation error has been identified (step 730). Returning to step 715, if L2 has not yet resolved C, then a dummy entry for C is created in the internal cache of L2 (step ~~840~~<u>740</u>), with the constrained value field in this entry being set to the class object reference obtained by L1 (step 745). The method then exits (step

745). Going back to step 710, if L1 has not already resolved C, a test is performed (step 750) to see if L2 has already resolved C. If not, dummy entries for C are created in the internal caches for both L1 and L2 (step 755), and the constraint parent field in the internal cache for L1 is set to point to the dummy entry for C in the cache for L2 (step 760). The method then exits (step 745).

Please replace the paragraph beginning at page 52, line 5, with the following rewritten paragraph:

An ~~object-oriented~~ A computer system includes multiple ~~a two or more~~ class loaders for loading program class files into the system. A constraint checking mechanism is provided ~~so that where~~ wherein a first class file loaded by a first class loader makes a symbolic reference to a second class file loaded by a second class loader, ~~with said~~ the symbolic reference including a descriptor of a third class file. ~~, the~~ The constraint mechanism ~~enforces~~ requires that the first and second class files agree on the identity of the third class file. ~~The constraint checking mechanism~~ and stores a list of constraints as a set of asymmetric relationships between class loaders. Each stored constraint, for a class loader which loaded a class file that contains a symbolic reference to another class file, includes a first parameter denoting the class loader which loaded the class file to which the symbolic references is made; and a second parameter denoting a class file which is identified by a descriptor in said symbolic reference.

4